Gates & Cooper LLP
Howard Hughes Center
6701 Center Drive West, Suite 1050
Los Angeles, California 90045

FAX TRANSMISSION TO USPTO

TO: Commissioner for Patents
MS APPEAL BRIEF - PATENTS
Patent Examining Corps
Facsimile Center
Alexandria, VA 22313-1450

FROM: Victor G. Cooper
OUR REF.: ST9-98-107US2
TELEPHONE: (310) 642-4142

Total pages, including cover letter: 17

PTO FAX NUMBER: 703/872-9306

If you do NOT receive all of the pages, please telephone us at (310) 641-8797, or fax us at (310) 641-8798.

Title of Document Transmitted:	**BRIEF OF APPELLANT. Please charge Deposit Account No. 09-0460 in the amount of $340.00 for the Appeal Brief filing fee.**
Applicant:	Thomas J. Pavela
Serial No.:	09/955,804
Filed:	September 19, 2001
Group Art Unit:	2122
Title:	SYSTEM AND METHOD FOR DEVELOPING TEST CASES USING A TEST OBJECT LIBRARY
Our Ref. No.:	ST9-98-107US2

Please charge all fees to Deposit Account No. 09-0460 of IBM Corporation, the assignee of the present application.

By: _____
Name: Victor G. Cooper
Reg. No.: 39,641

I hereby certify that this paper is being transmitted by facsimile to the U.S. Patent and Trademark Office on the date shown below.

Isabel Ogata
Signature

Nov. 24, 2004
Date

VGC/io

G&C 30571.210-US-C1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:	
Inventor: Thomas J. Pavela	Examiner: Vo, Ted T.
Serial #: 09/955,804	Group Art Unit: 2122
Filed: September 19, 2001	Appeal No.: _____
Title: SYSTEM AND METHOD FOR DEVELOPING TEST CASES USING A TEST OBJECT LIBRARY	

BRIEF OF APPELLANT

MAIL STOP APPEAL BRIEF - PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

In accordance with 37 CFR §1.192, Appellant hereby submits the Appellant's Brief on Appeal from the final rejection in the above-identified application, as set forth in the Office Action dated September 24, 2004.

Please charge the amount of $340 to cover the required fee for filing this Appeal Brief as set forth under 37 CFR §1.17(c) to Deposit Account No. 09-0460 of I.B.M. Corporation, the assignee of the present application. Also, please charge any additional fees or credit any overpayments to Deposit Account No. 09-0460.

I. REAL PARTY IN INTEREST

The real party in interest is the IBM Corporation, the assignee of the present application.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences for the above-referenced patent application.

III. STATUS OF CLAIMS

Claims 22-36 are pending in the application.

Claims 22-24, 26-29, 31-34, and 36 were rejected under 35 U.S.C. §102(a) as being unpatentable over U.S. Patent No. 5,754,755, issued to Smith (hereinafter the Smith reference) and these rejections are being appealed.

Claims 25, 30, and 35 were rejected under 35 U.S.C. §103(a) as being obvious over Smith.

IV. STATUS OF AMENDMENTS

No amendments to the claims have been made subsequent to the final Office Action.

V. SUMMARY OF THE INVENTION

Briefly, Appellant's invention, as recited in independent claims 22, 27,and 32, is described as a method, apparatus, and article of manufacture, for generating a test code for an automatic procedure. As disclosed in FIGs. 2 and 3 and on page 7, line 25 to page 8, line 5 of the Applicants' specification, the method comprises the steps of defining a source file having a plurality of tags associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure, generating a test plan in a conventional language from the source file, and generating an automated test code for the automated test procedure from the source file. An exemplary source file with tags is presented in FIG. 4 and the text related thereto (page 8, lines 11 - page 9, line 2. In one embodiment, a test index (page 10, lines 19-23) identifying system elements tested by the test code is generated and incorporated into the test plan, allowing the user to verify that all desired system elements are exercised by the automated test code. Automated test code is generated using a technique wherein commands to the system elements are issued and messages responsive to the commands are intercepted and used to provide test status and error messages (page 11, lines 8-21). The article of manufacture comprises a data storage device tangibly embodying instructions to perform the method steps described above.

VI. ISSUES PRESENTED FOR REVIEW

Whether claims 22-24, 26-29, 31-34, and 36 are patentable under 35 U.S.C. § 102(a) over the Smith reference.

Whether claims 25, 30, and 35 are patentable under 35 U.S.C. § 103(a) over the Smith reference.

VII. GROUPING OF CLAIMS

The rejected claims do not stand or fall together. Each claim is independently patentable. Separate arguments for the patentability of the following claim groups are presented:

Claims 22 and 27;

Claims 24, 29, and 34;

Claims 26, 31, and 36; and

Claims 25, 30, and 35.

VIII. ARGUMENTS

A. Claims 22-24, 26-29, 31-34, and 36 Are Patentable Over The Prior Art

1. The Smith Reference

U.S. Patent No. 5,754,755, issued May 19, 1998 to Smith, discloses a method and system for generating an application-specific test script file. The application-specific test script file contains test instructions for testing an application program. The system receives a test template file that has test instructions that contain placeholders. The placeholders indicate where application-specific placeholder values are to be logically inserted into the test template file. The system receives an ordered list of customizing files that have application-specific placeholder values. The system then searches the customizing files according to the ordered list for a first placeholder value for each placeholder of the test instruction. When such a placeholder value is found, the system replaces the placeholder with the placeholder value in the test instruction and stores the test instruction into the application-specific test script file.

2. Claim 22 is Patentable Over the Smith Reference Under 35 U.S.C. § 102(a)

Claim 22 recites:

A method of generating test code for an automated test procedure applyable to a system comprising a plurality of interconnected elements, the method comprising the steps of:

defining a source file having a plurality of tags, each tag associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure;

generating a test plan in a conversational language from the source file; and

generating the test code for the automated test procedure from the source file.

According to the First Office Action, Smith discloses "defining a source file having a plurality of tags," in elements 112 and 144 of FIG. 1:

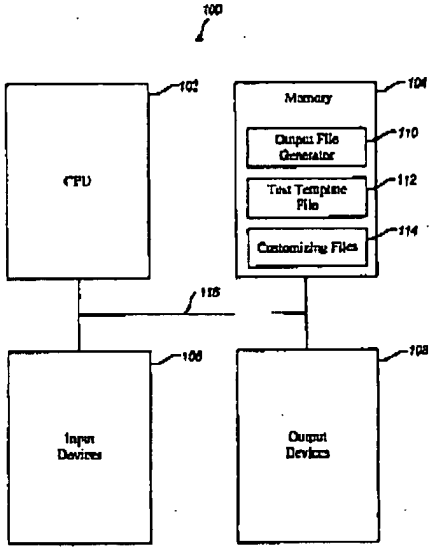

Fig. 1

and in the text cited below:

> For example, the HTML template file might contain placeholders for stock price and volume data. The customizing files stored in the server contain the current price and volume information that would be inserted into the HTML output file (application-specific test script). (col. 7, lines 25-30)

Essentially, the First Office Action argued that the "source file" of the Applicant's invention is analogous to the test template file 112 of the Smith reference, and that the "tags" of the Applicant's invention are analogous to the "placeholders".

Referring to the following passage, the First Office Action further suggested that each placeholder is "associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure":

> The ##FOR command indicates that iteration statements follow. The ##NEXT statement indicates the end of the iteration statements that are to be repeated over the range of numerical values from a to b. For each iteration of the iterative command, the indexed placeholder within the iteration statements is replaced by a placeholder with it's asterisk (*) by a value starting with the value of a and sequentially increasing to the value of b. For example, for each iteration of the iterative commands 210, 213, the placeholder takes on the value OBJECT1, OBJECT2, OBJECT3, etc. The output file generator searches the customizing files for a values for the generated placeholders, replaces the generated placeholders with those values, and stores the instructions 211, 212 in the application-specific test script. In addition, "a" and "b" may be placeholders, in which case, their value would be retrieved from a customizing file prior to the evaluation of the #FOR command. (col. 5, lines 49-65).

The Applicant responded by pointing out that the "placeholders" were not analogous to the "tags" of the Applicant's invention, because the were not associated with a library of executable code objects defining a set of instructions for performing a portion of the automated test procedure, as recited in claim 22:

> However, Smith's "placeholders" are not analogous to the "tags" of the Applicant's invention, because the "placeholders" are not *associated with a library of executable code objects (314) defining a set of instructions for performing a portion of the automated test procedure.* Instead, Smith's "placeholders" are precisely that ... placeholders that are *replaced* with application specific *data* (which are essentially arguments) that are obtained from the customizing files (see col. 3, lines 11-13). The placeholders are not associated with a library of executable code objects defining a set of instructions.

And provided evidence that Smith's placeholders were used to substitute data values, not one of a library of executable code objects defining a set of instructions for performing a portion of the automated test procedure.

> "Unlike the Smith reference which teaches the substitution of *placeholders* with *data values* from a customizing file, the Applicant's invention teaches the use of tags which are associated with a library of executable code objects defining a set of *instructions* for performing a portion of the automated test procedure."

The Applicant also remarked that the difference between claim 22 and Smith was not surprising, given that they are directed to different purposes:

> "The foregoing differences are a product of the different purposes to which they are directed. The Smith reference offers a testing mechanism that permits use of a single test script for different application programs, but allows for differences between the programs (see col. 2, lines 19-22). Thus, it provides a template that can be used with multiple application programs and includes placeholders that are replaced with different input values (each for a different application program). The Applicant's invention is not directed to creating a test program that is applicable to different applications, but rather, a system that guides generation of test cases, and thus, provides tags associated with a library of executable code objects that can be used to debug the code."

The Final Office Action disagreed:

> "It is noted that placeholder is used with HTML tags as addressed by Smith (col. 7, lines 26-30). To interact with library functions, it requires a value to be placed in the placeholder. For example, at column 4, lines 15-20, a name of a function such as *function2* (claimed limitation: a member of a library of executable code objects) is placed in the placeholder; a command "*execute function 2*" invokes this function for execution. The reference of Smith associates with HTML tags. Furthermore, right in the column 2, lines 45-58, Smith states "*the output generator receives a test template file that has statements that are test instructions or iterative commands, and that has place holders*". Thus, test instructions, or iterative commands generated by Output File Generator 110, not by a user; therefore, *test instructions* and *iterative commands* also have a means of "*a member of a library of executable code objects defining a set of instructions for performing a portion of automatic test procedure.*" (page 3, emphasis in original)

The Examiner argues that the following passage discloses the disputed phrase of claim 22:

> The output file generator also can dynamically generate placeholders for a test script. For example, a tester may want to define a series of placeholders in a test template file with similar names. **Such a tester may want to test 10 different functions of the application program with similar instructions except with different function names depending on the application program.** The output file generator allows the tester to specify an iterative control command with an index and iterative statements with an indexed placeholder. The iterative statements can be instructions or control commands. For each iteration, the output file generator stores the iterative in the test script with the indexed placeholder replaced with a placeholder value from a customizing file. For example, the following is an example of an iterative command:
>
> ```
> ##FOR[{Function*}]= 1 to 10
>         execute function*
> ##Next
> ```
>
> The iterative control command is the ##FOR/##NEXT statement and the iterative statement is the instruction "execute function*" where "*" is the index and **"function*" is the indexed placeholder.** The output file generator processes the iterative control command as if the instructions "execute function1," "execute function2," and so on to wherein the template test script. (col. 4, lines 1-29)

In an Amendment under 37 C.F.R. 1.116, the Applicant stated:

"However, the 'functions' referred to above are functions of the tested application program. In other words, if the application program includes *function1, function2, ... function10*, the foregoing permits all 10 functions of the program to be iteratively tested, and it does so by allowing the use of an indexed 'placeholder' called *function**.

These functions, however, are **part of the application program to be tested,** not test code, and are therefore not *associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure*, as claim 22 recites.

Smith teaches substituting functions of the application program itself into a test script. It does not teach substituting *test procedure* executable code objects into a source file that is used to generate a test code. In fact, as the foregoing makes clear, Smith, like the rest of the prior art the Applicant is aware of, teaches hardcoding such test procedures, and in doing so, teaches away from the Applicant's invention."

The Advisory Action responded:

"Applicants have not been able to explain why, HTML template file containing placeholders are not analogous to 'tags' as given in the Examiner's citation in Final action: dated 6/01/04: page 2, section 2:

> 'The output file generator of the present invention generates a Web page using user independent WEB page template file and the customizing data files. For example, the HTML template file might contain a placeholder for stock price and volume data' (Addressed to limitation TAGS).'

It is known that HTML template file is a tag script file. According to Smith, 'placeholders' are used to take values of OBJECT1, OBJECT 2, OBJECT3. Thus, 'placeholders' are areas in the WEB page template file used by a user to fill the values of OBJECT1, OBJECT2, OBJECT3 (addressed to limitation: associated with a member of a library of executable code objects).

In a particularly case, Smith uses HTML to write testplan, where testplan simply a text of source/document like what it can be read from a web site document."

The Applicant's answer is simple ... OBJECT1, OBJECT2, and OBJECT3 are not *members of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure*, as recited in claim 22. Perhaps the confusion is best clarified by considering a specific of OBJECT1, OBJECT2, and OBJECT3. As shown in FIG. 3B, "Microsoft Word Document", "Microsoft Word

Picture", and "Microsoft Powerpoint Slide" are not *members of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure.*

For the foregoing reasons, the Applicant respectfully traverses the rejection of claim 22. Claim 27 recites analogous features, and the rejection of this claim is traversed on the same basis.

3. Claims 24, 29, and 34 are Patentable Over the Smith Reference Under 35 U.S.C. § 102(a)

Claim 24 recites:

The method of claim 23, wherein the test plan comprises a test index identifying the system elements tested by the test code, the test index generated by performing the step of scanning the interpreted tags to identify the system elements tested by the test code.

According to the First Office Action, the Smith reference discloses the generation of a text index as follows:

> When generating an application-specific test script, the output file generator scans through the customizing files in order searching for a value for each placeholder. (col. 3, lines 45-48)

The Applicant responded:

"However, this does not disclose the generation of a 'test index identifying system elements tested by the test code' nor does it disclose performing this task by 'scanning the *interpreted* tags to identify the system elements tested by the test code,' as recited in claim 22."

The Final Office Action explains:

"'Scanning' has means for identifying. Particularly, Smith shows scanning means (column 3, lines 45-48). Although the scanning is for customizing file for searching values of place holder, this teaching includes scanning test instructions or iterative commands (See column 2, lines 45-58), HTL templates (Column 7, lines 10-53) that contain tags and placeholders."

"Smith teaching includes HTML template with placeholders, and HTML tags for generating documents and test plans. HTML tags provide identifying elements associated with it. As discussed above for 'Function*', Smith refers to the symbol '*' as indexing that causes a function with a particular index would be executed. For example, 'execute function2' (*test index identifying system elements tested by the test code*) (See Column 4, lines 22-28). And with the scanning means is used by Smith, it is obvious that scanning would interpret the tags in the HTML template filed that associated with 'function2'." (Final Office Action, page 4)

As a threshold matter, the foregoing argument appears to reject the claim as "obvious" over what Smith "discloses." The Applicant respectfully points out that this claim was rejected under 35 U.S.C. § 102(a), not 35 U.S.C. § 103, and statements regarding the "obviousness" of what one might do using the Smith reference are of questionable relevance.

Further, the foregoing argument is clearly based on hindsight reconstruction, and taking single words out of the context of the entire phrase. Smith discloses an index, "*". However, that "index" is not a test index ... it is an index used in an iterative process. The "*" does not identify system elements tested by the test code, nor is it generated by scanning the interpreted tags to identify system elements identified by the test code. Any argument to the contrary is an exercise in hindsight reconstruction.

Claims 29 and 34 recite the features of claim 24 and are patentable on the same basis.

4. Claims 26, 31, and 36 are Patentable over the Smith Reference Under 35 U.S.C. § 102(a)

Claim 26 recites:

The method of claim 22, wherein the step of generating test code for the automated test procedure comprises the step of translating the executable code objects associated with the tag in the source file.

As described above, Smith does not disclose tags associated with executable code objects defining a set of instructions for performing a portion of the automatic test procedure, and therefore cannot disclose translating the executable code objects associated with tags. Accordingly, the Applicant traverses the rejection of claim 26. Claims 31 and 36 recite features analogous to those of claim 26 and are patentable for the same reasons.

B. Claims 25, 30, and 35 Are Patentable Over The Prior Art Under 35 U.S.C. § 103

Claim 25 recites:

The method of claim 23, wherein the step of generating a test plan further comprises the steps of:
identifying an uninterpretable tag in the test plan; and
appending the test plan with an error message identifying the uninterpretable tag.

The First Office Action argued that the foregoing is "syntax" error checking, and that such checking is well known in the art. The Applicant responded:

"However, recalling that in rejecting claim 22, the Office Action analogized 'tags' to 'placeholders', which, as described above, is a placeholder for *data*, and *data* is not 'uninterpretable.'

The foregoing difficulty, of course, is a product of the erroneous premise that the Applicant's 'tags' are analogous to Smith's 'placeholders.' Nonetheless, since one of ordinary skill in the art would not perform a syntax error check on a placeholder for 'data,' the Applicant traverses this rejection."

The Final Office Action respectfully responded:

"As noted above, while the prior action citation includes HTML tags, Applicants are silent on HTML tags as cited in the previous Office Action in regard to Claim 22. In the previous Office Action, it was not associated syntax error to check a *placeholder*, but addressed to syntax error and issuing the error as well known features associated in all type of programming in dealing with syntax and semantic checking. The ordinarily skilled in the arts would know that the syntax check is performed in test instructions, iterative commands, and tags of the HTML script." (Final Office Action, page 5)

The only paragraph in the Smith reference that includes the acronym "HTML" is presented below:

> In an alternative embodiment, the present invention is used to generate a page on the WORLD WIDE WEB ("WEB page"). A WEB page typically includes graphics and is coded using hypertext markup language ("HTML"). Usually, when a user wishes to view a WEB page, the user requests that it be downloaded from a server, on which it resides, onto a local computer. Because the WEB page typically includes graphics, the downloading is time-consuming. Using the present invention, a template file for a WEB page is stored on the local computer, and customizing files are stored on the server. **When a user wishes to view a WEB page, to save time, only the customizing files are downloaded. Then, the output file generator of the present invention generates a WEB page using the user-independent WEB page template file and the customizing data files. For example, the HTML template file might contain placeholders for stock price and volume data. The customizing files stored on the server contain the current price and volume information that would be inserted into the HTML output file** (application-specific test script). (col. 7, lines 11-30, emphasis added)

The foregoing appears to describe an embodiment that has nothing whatever to do with the generation of test cases or even testing software. Instead, it refers to the use of the "customizing file" paradigm to promote quicker downloading of web pages to users.

Nothing in the Smith reference teaches the generation of a test plan *which has conversational language phrases for each translated tag.* Nothing in the Smith reference discloses identifying an uninterpretable tag in a test plan, and nothing discloses appending the test plan with an error message identifying the uninterpretable tag. Accordingly, the Applicant respectfully traverses this rejection.

Claims 30 and 35 recite limitations analogous to those of claim 25, and are patentable for the same reasons.

IX. CONCLUSION

In light of the above arguments, Appellant respectfully submits that the cited references do not anticipate nor render obvious the claimed invention. More specifically, Appellant's claims recite novel physical features, which patentably distinguish over any and all references under 35 U.S.C. §§ 102 and 103. As a result, a decision by the Board of Patent Appeals and Interferences reversing the Examiner and directing allowance of the pending claims in the subject application is respectfully solicited.

Respectfully submitted,

GATES & COOPER LLP

Attorneys for Applicant

Howard Hughes Center
6701 Center Drive West, Suite 1050
Los Angeles, California 90045
(310) 641-8797

Date: November 24, 2004

By: _____
Name: Victor G. Cooper
Reg. No.: 39,641

VGC/io

G&C 30571.210-US-C1

APPENDIX

1-21. (CANCELED)

22. (PREVIOUSLY PRESENTED) A method of generating test code for an automated test procedure applyable to a system comprising a plurality of interconnected elements, the method comprising the steps of:

defining a source file having a plurality of tags, each tag associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure;

generating a test plan in a conversational language from the source file; and

generating the test code for the automated test procedure from the source file.

23. (PREVIOUSLY PRESENTED) The method of claim 22, wherein the step of generating a test plan comprises the steps of:

translating the tags; and

generating a conversational language phrase for each translated tag.

24. (PREVIOUSLY PRESENTED) The method of claim 23, wherein the test plan comprises a test index identifying the system elements tested by the test code, the test index generated by performing the step of scanning the interpreted tags to identify the system elements tested by the test code.

25. (PREVIOUSLY PRESENTED) The method of claim 23, wherein the step of generating a test plan further comprises the steps of:

identifying an uninterpretable tag in the test plan; and

appending the test plan with an error message identifying the uninterpretable tag.

26. (PREVIOUSLY PRESENTED) The method of claim 22, wherein the step of generating test code for the automated test procedure comprises the step of translating the executable code objects associated with the tag in the source file.

27. (PREVIOUSLY PRESENTED) An apparatus for generating test code for an automated test procedure applyable to a system comprising a plurality of interconnected elements, comprising:

means for defining a source file having a plurality of tags, each tag associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure;

means for generating a test plan in a conversational language from the source file; and

means for generating the test code for the automated test procedure from the source file.

28. (PREVIOUSLY PRESENTED) The apparatus of claim 27, wherein the means for generating a test plan comprises:

means for translating the tags; and

means for generating a conversational language phrase for each translated tag.

29. (PREVIOUSLY PRESENTED) The apparatus of claim 28, wherein the test plan comprises a test index identifying the system elements tested by the test code, wherein the test index generated by performing the step of scanning the interpreted tags to identify the system elements tested by the test code.

30. (PREVIOUSLY PRESENTED) The apparatus of claim 28, wherein the means for generating a test plan further comprises:

means for identifying an uninterpretable tag in the test plan; and

means for appending the test plan with an error message identifying the uninterpretable tag.

31. (PREVIOUSLY PRESENTED) The apparatus of claim 27, wherein the means for generating test code for the automated test procedure comprises means for translating the executable code objects associated with the tag in the source file.

32. (PREVIOUSLY PRESENTED) A program storage device, readable by a computer, tangibly embodying at least one program of instructions executable by the computer to perform

method steps of generating test code for an automated test procedure applyable to a system comprising a plurality of interconnected elements, the method comprising the steps of:

defining a source file having a plurality of tags, each tag associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure;

generating a test plan in a conversational language from the source file; and

generating the test code for the automated test procedure from the source file.

33. (PREVIOUSLY PRESENTED) The program storage device of claim 32, wherein the method step of generating a test plan comprises the method steps of:

translating the tags; and

generating a conversational language phrase for each translated tag.

34. (PREVIOUSLY PRESENTED) The program storage device of claim 33, wherein the test plan comprises a test index identifying the system elements tested by the test code, the test index generated by performing the step of scanning the interpreted tags to identify the system elements tested by the test code.

35. (PREVIOUSLY PRESENTED) The program storage device of claim 33, wherein the step of generating a test plan further comprises the method steps of:

identifying an uninterpretable tag in the test plan; and

appending the test plan with an error message identifying the uninterpretable tag.

36. (PREVIOUSLY PRESENTED) The program storage device of claim 32, wherein the method step of generating test code for the automated test procedure comprises the method step of translating the executable code objects associated with the tag in the source file.